CERN SL Note 97-55 (AP)

Basic MAD Tables in *Mathematica*

John M. Jowett
18/8/1997, Revised 1/10/1999

1 Introduction

This note is the documentation for a *Mathematica* [1,2] package designed to handle data in a form typical of a class of table files. These include the TFS files generated by certain MAD [3] commands (TRACK, EIGEN, TWISS3, OPTICS, etc., possibly via MAD's ARCHIVE command) and a number of different types of files saved by application programs in the LEP control system. Although some of these do not strictly adhere to the TFS conventions, their data is easily assimilated into a common data type **`mfs`** in *Mathematica*.

Although not very interesting in itself, this package can be used in many ways. Moreover it provides a basis for a number of other application packages (such as **`TrackTable`**) that extend its functionality in various directions.

These packages will allow you to operate *directly* with physically interesting quantities (eigenvectors, orbits, transfer matrices, maps, etc.) and manipulate them in natural mathematical style, without having to worry about the technicalities of file and data formats. While MAD itself provides an excellent framework for many kinds of beam optics and particle tracking calculations, I think it is fair to say that it's capabilities are still somewhat under-exploited because of the lack of tools for making sense of the abundance of results it can produce.

There have been a number of attempts to fill this gap in recent years. Personally I have found that the environment provided by *Mathematica* is particularly fruitful and easy to work in. However my experience has shown that the speed with which applications can be developed tends to leave documentation far behind. This series of packages, developed in collaboration with K. Goral, is being organised systematically according to established guidelines for package development. This should make them robust and easy to use and maintain.

These packages and others relating to the use of MAD will all be made available in the context **`Madtomma`** (the word *context* is used in a special sense in *Mathematica).*

The package has been thoroughly tested with tables from both MAD Version 8 and Version 9.

A useful feature is a *palette* of buttons that frees the user from most of the need to remember and type in the syntax of the functions in the package. See Section 8 below.

Access to the packages has been simplified thanks to changes in the CERN computer systems and the "User's Guide and Examples" below have been enhanced since the first version of the package.

It is very likely that further functions will be added in future versions of the package. Therefore it is always worth checking the latest version of this notebook at the Madtomma Web site

```
http://wwwslap.cern.ch/jowett/Madtomma
```

This notebook adheres to the conventions for *Mathematica* package structure and documentation set out in [2]. As such it serves as the development medium for the package itself. **For this reason , some sections of the printed**

versions of this document are hidden. They contain the definitions ("code") of the functions. The visible sections contain the documentation and examples of interest to users.

To make the package available, you need only copy a couple of commands from the "Setup" section. The "Examples" section illustrates basic use of the package and is the best place to start learning how to use the package. More elaborate "real-life" examples (e.g. applications to the LHC) can be found at the Web site.

A final subsection illustrating some less useful features is hidden.

This notebook is based on the template for package and notebook development by R. Maeder and supplied with *Mathematica* 3.0.

1.1 New in version 2.0

The names of several functions have been changed to make them more consistent and easier to guess or remember,The old function names are still accepted but generate warning messages. A function **`mfsVersion2Update`** has been provided to facilitate the updating of existing files.

The package is now compatible with MAD Versions 8 and 9.

New functions have been added to allow different TFS tables to be merged with checking that the merging is valid.

2 Reference

2.0.1 Title

Basic MAD Tables in *Mathematica*

2.0.2 Author

John M. Jowett

2.0.3 Summary

This notebook provides functions for creating and manipulating the mfs data type created by various other packages. In particular it creates mfs data itself from TFS data files saved by MAD and other programs.

This package provides a basis for other packages that will be dependent upon it.

2.0.4 Copyright

2.0.5 Notebook Version

2.2

2.0.6 *Mathematica* Version

3.0

2.0.7 History

Starting from ReadTrackTable.nb 12/6/97.
Added removeQuotes function to fix a bug, 13/8/97.
Added colValue, mfsSelect, mfsMemebr, mfsRange, mfsReverse functions 15/8/97.
Added keepTemporaryFile option to tfsRead, improved the setting of $Path and finding the example file 24/4/98.
Modified mfsRange to include the endpoints of the interval (< replaced by ≤, etc.), 18/9/98.
Added handling of %le format obtained when OPTION,DOUBLE is used in MAD, 21/9/98.
Addition of palette, simplification of Setup section, etc. 3/2/1999.
Added %s format specifier used by MAD9 TFS output, 6/9/1999.
Modified tfsParseHeaderBlock to allow arbitrary order of header lines for MAD9, 7/9/1999.

Version 2.0 created with major changes, 13/9/1999.
The main motivation is compatibility with MAD9 but other improvements have been mae.
Several functions have been given new, more logical names, beginning "mfs...". The list of changes that need to be made in any existing files is given as **`mfsVersion2UpdateRules`** and a function **`mfsVersion2Update`** has been provided to facilitate the updating of files. The old names of functions will continue to work but generate an warning message. All cells containing code to facilitate the transition are coloured red in this notebook. They may eventually be removed.
The function **`mfsMerge`** has been created to allow mfs objects to be merged.
Some other editing functions also added.

Version 2.1 added the **`stringRemoveLeadingTrailingBlanks`** function so that makeEigenTable etc. work for MAD Version 9.
Version 2.2 22/3/2000, added change of name of symplecticJ here so as to take care of today's update to TrackTable package.

2.0.8 Keywords

TFS, MAD, data, table, OPTICS, TRACK, mfs

2.0.9 Source

For a brief description of the TFS file format as used in MAD, see Appendix C of H. Grote, F.C. Iselin, "The MAD Program, Version 8.16, User's Reference Manual", CERN/SL/90-13 (AP) 1995, or

`http://wwwslap.cern.ch/fci/mad/mad8/tfs.html`

2.0.10 Warnings

Note: all cells marked as "InitializationCell" will be written to the Auto-Save package. This package can then be read in programs that use it with **`Needs["Madtomma`Mfs`Mfs`"]`**. Cells not intended to belong to the package should not have this property.

2.0.11 Discussion

Here we provide the internal specification of **`mfs`** data objects. Normal users may skip it.

Basically an mfs data object is a list with head **`mfs`** and three or more elements.

The *first* element is a list of keys. Each key consists of a descriptive string, usually in uppercase, such as **`"QX"`** and a corresponding value, such as **`96.24`**. In TFS data files the value is either a number or a string. In mfs data, it can be any kind of object definable in *Mathematica*, e.g., a matrix consisting of numerical and symbolic elements.

The *last* element is a table, consisting of columns of data. If the data comes from a TFS file, then the elements of a given column are all or the same type, either numbers or strings.

The *second-last* element is a set of string labels, one for each of the columns in the last part.

Additional parts between the first and the second-last may be used for other purposes.

Note that the first element of an mfs data object is actually an **associative array** in the sense used in various programming languages. You can also think of the second-last and last elements combined as another associative array with the constraint that the values are all numerical lists of the same length. This is necessary because there are cases where the key "X", for example, occurs both in the header block (as a closed-orbit component) and in the column names (as the name of a phase-space coordinate).

If you like the **object-oriented programming** metaphor, you can think of **`mfs`** as a class and various functions in this package as the associated methods.

2.0.12 Requirements

Uses a standard package for column manipulations, etc.

Statistics`DataManipulation`

The functions in this package are available once this one is loaded (and are often very useful - see the documentation for this package).

3 Setup

This section contains commands needed to load the corresponding package file. The contents of this file are equivalent to the following sections (Interface, Implementation, Epilog) in which the package is developed.

3.1 Search Path (ESSENTIAL!)

To have access to my packages, you may need to add my packages directory to your search path. This is system-dependent and the latest information about arranging it on CERN computer systems can be found at

`http://wwwslap.cern.ch/~jowett/Madtomma/AboutFiles.html`

and is not reproduced here. I strongly recommend that you modify your kernel initialisation file once and for all as explained on this page. Then all my packages will be found as easily as the Standard Packages that come with *Mathematica*.

3.2 Loading the Package

Once the package directory is on your search path, the **`Mfs`** package can be loaded by evaluating the following cell. *If you are using a copy of this notebook in order to work through the examples and you are invited to evaluate all the initialisation cells in it, you should click "NO" and then go straight to the "User's Guide and Examples" without evaluating the intermediate sections.*

```
Needs["Madtomma‘Mfs‘Mfs‘"]
```

```
Version 2.0 of Madtomma‘Mfs‘Mfs‘ loaded.
   Note function name changes since Version 1.x:

interpretDescriptors → mfsInterpretKeys
keyValue → mfsKeyValue
columnNames → mfsColumnNames
descriptorNames → mfsKeyNames
colValue → mfsColumnValue
addDescriptor → mfsAddKey
editDescriptor → mfsEditKey
deleteDescriptor → mfsDeleteKey
```

This is all you need to start using the package in your own applications. In interactive sessions, you will normally see a palette of buttons appearing on your screen. This makes it easier to use the package: see Section 8 below.

These following sections are hidden when this notebook is used as the package documentation but may be inspected in the online copy that can be found in the appropriate sub-directory of the directory added to the **$Path** variable above.

4 Interface

5 Implementation

6 Epilogue

7 User’s Guide and Examples

7.1 Normal usage of the package

7.1.1 Creating an mfs object from a file of data

First, let us define a sample data file. This one was created in MAD using the ARCHIVE command to save the TRACK table generated by tracking a number of particles from the same initial condition with quantum fluctuations.

```
sampleFile = ToFileName[{AfsHomeDirectory["jowett"], "public",
   "math", "Madtomma", "Mfs", "Examples"}, "qkick.tfs"]
P:\cern.ch\user\j\jowett\
  public\math\Madtomma\Mfs\Examples\qkick.tfs
```

Choose some directory of your own for the examples

```
SetDirectory["C:\\temp"]
C:\temp
```

Having loaded the package, we can list all the data types in the **mfs** class by evaluating:

```
mfsTypes
{mfs}
```

The most common applications will start by reading a TFS file:

```
?tfsRead
```

```
tfsRead[file] returns an mfs data object
  containing all the information in a TFS file.
```

In any expression which evaluates to an mfs data object, it is worth remembering to add a semi-colon to suppress printing as these can be very long.

```
qpmfs = tfsRead[sampleFile, Verbose -> False];
```

MAD users know that this leads to a file containing various pieces of descriptive information at the top, followed by a body of columns listing the coordinates of the surviving particles on successive turns, separated by comment lines.

Since only one of the particles survives the full tracking time, the others being lost after varying numbers of turns, the number of lines of the TFS file corresponding to each turn gradually dwindles. The **tfsRead** function deals with these details automatically.

Now the symbols **qpmfs** contains an mfs data object which we can print in abbreviated form:

```
Short[qpmfs, 8]
mfs[{{TYPE, TRACK}, {X, -0.000388354},
  {PX, -0.0000167029}, {Y, -0.000381844}, {PY, -2.94935×10^-6},
  {ET, 0.0000102626}, {EY, 1.05428×10^-10}, {EX, 2.98257×10^-8},
  {E11, 5.25333002895}, <<31>>, {E63, -0.000117069},
  {E64, 0.000342355}, {E65, 0.00400956}, {E66, 0.420458776176},
  {COMMENT, }, {ORIGIN, MAD 8.21/11 RS6000 - AIX},
  {DATE, 01/05/97}, {TIME, 15.40.36}}, <<1>>, { <<1>> }]
```

The option **Verbose->False** can be used to suppress the printed information in **tfsRead** and other functiions.

7.1.2 Extracting header information from an mfs object

In this example, the TFS file contained a TRACK table from MAD. The header block of the file contains various descriptors that are converted into *keys* and associated *values* in the mfs data object. To find out what they are we can evaluate

```
mfsKeyNames[qpmfs]
{TYPE, X, PX, Y, PY, ET, EY, EX, E11, E12, E13, E14, E15, E16,
 E21, E22, E23, E24, E25, E26, E31, E32, E33, E34, E35, E36,
 E41, E42, E43, E44, E45, E46, E51, E52, E53, E54, E55, E56,
 E61, E62, E63, E64, E65, E66, COMMENT, ORIGIN, DATE, TIME}
```

The function **mfsKeyValue** can be used to extract the values corresponding to the keys:

```
pxc = mfsKeyValue[qpmfs, "EX"]
2.98257 × 10^-8
```

```
mfsKeyValue[qpmfs, "ORIGIN"]
MAD 8.21/11 RS6000 - AIX
```

If a key does not exist, an error message will be given and a **Null** value returned.

```
mfsKeyValue[qpmfs, "QX"]
```

– *mfsKeyValue::notfound : Descriptor QX not found.*

Lists can be extracted with one call, e.g, to get four components of the closed orbit

```
mfsKeyValue[qpmfs, {"X", "PX", "Y", "PY"}]
{-0.000388354, -0.0000167029, -0.000381844, -2.94935 × 10^-6}
```

Of course, lists can be used in many ways in *Mathematica*.

```
ArcTan @@ mfsKeyValue[qpmfs, {"X", "PX"}]
-3.09861
```

7.1.3 Extracting column information from an mfs object

The body of the original TFS file contained columns of data. To find out what these are we use:

```
mfsColumnNames[qpmfs]
{TURNS, PARTICLE, X, PX, Y, PY, T, DELTAP}
```

To extract columns of data we can use the function **mfsColumn**

```
?mfsColumn
```

```
mfsColumn[mfsdata,colname] returns the column of
  data labelled by the string colname in an mfs (or
  related) data object. A list of colnames may also
  be given to return a set of columns.  If colname
  is absent the entire block of columns is returned.
```

Thus, for example, we can get a list of all the values of the momentum PY. Since this is long, we abbreviate the printing with the function **Short:**

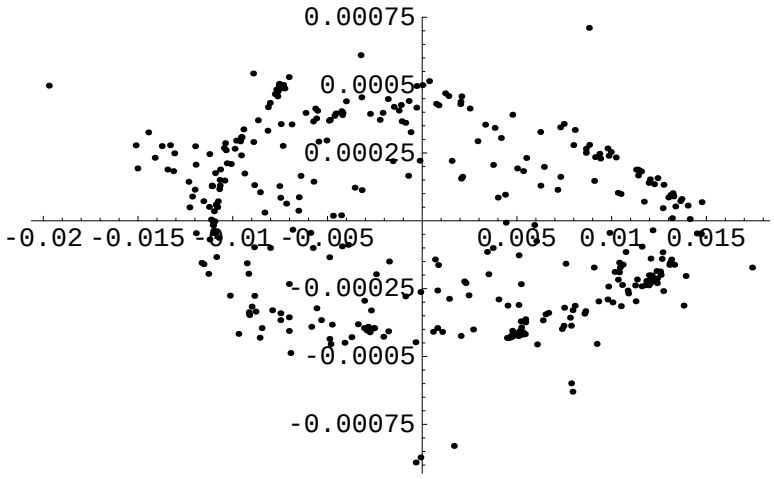
```
Short[ mfsColumn[qpmfs, "PY"] , 8]
{0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 <<348>>, -0.0000369869, 0.0000155165, 0.000038503, 0.00001693,
 -0.00001942, -0.0000366784, -0.0000171618, 0.0000266039,
 0.0000343572, -3.51693×10^-6, -0.0000327806, -0.0000123483,
 0.000017505, 0.0000320125, 0.0000178347, -0.0000214404}
```

Multiple columns can also be returned. Using **Transpose**, these can be transformed into lists of coordinate vectors. Here are the coordinate pairs in the horizontal phase plane (abbreviated again)

```
Short[ xpx = Transpose[mfsColumn[qpmfs, {"X", "PX"}]] , 10]
{{0., 0.0005}, {0., 0.0005}, {0., 0.0005}, {0., 0.0005},
 {0., 0.0005}, {0., 0.0005}, {0., 0.0005}, {0., 0.0005},
 {0., 0.0005}, {0., 0.0005}, <<361>>, {0.00441111, -6.78022×10^-6},
 {-0.00175107, -0.000149943}, {-0.00319526, 0.000113019},
 {0.00398775, 0.0000855205}, {0.000672365, -0.000142523},
 {-0.00427998, 0.0000203029}, {0.00204503, 0.000155579},
 {0.00341965, -0.000114564}, {-0.0039068, -0.0000886172}}
```

which is just what is needed for the standard **ListPlot** function. Note that this example mixes the trajectories of all the particles in the file (the TrackTable package deals with this in a better way)

```
ListPlot[xpx]
```




```
- Graphics -
```

Non-existent columns return **Null**

```
mfsColumn[qpmfs, "PXN"]
```

```
mfsColumn::notfound : Column PXN not found.
```

Finally, calling **mfsColumn** with only one argument returns a list containing all the columns (the entire "body" of the original data file).

```
Short[mfsColumn[qpmfs], 10]
{{0, 1, 0., 0.0005, 0., 0., 0., 0.},
 {0, 2, 0., 0.0005, 0., 0., 0., 0.}, {0, 3, 0., 0.0005, 0., 0., 0., 0.},
 <<374>>, {106, 3, 0.00204503, 0.000155579,
  0.000287351, 0.0000320125, -0.00656549, -0.000508005},
 {107, 3, 0.00341965, -0.000114564, 0.000730013, 0.0000178347,
  -0.00252763, -0.00143335}, {108, 3, -0.0039068, -0.0000886172,
  0.000805955, -0.0000214404, 0.00293916, -0.00158017}}
```

The dimensions of this show us that there were 380 "lines" containing the 8 items specified by **mfsColumnNames[qpmfs] above.**

```
Dimensions[mfsColumn[qpmfs]]
{380, 8}
```

7.1.4 Calculations using column information from an mfs object

Suppose that we would like to form the product of all the X and PX values in the **mfs** object **qpmfs**.

In the plotting example above, it was necessary to apply **Transpose** to the result of **mfsColumn[qpmfs, {"X", "PX"}]** to transform the pair of long lists of X and PX values into a long list of pairs **{X,PX}** as required by **ListPlot**. Having done this, it is easy to form, say, the product of all the X and PX values by mapping an appropriate pure function over the list. Here is one which forms the product of X and PX

```
Short[(#1[[1]] #1[[2]] &) /@ xpx]
{0., 0., 0., <<375>>, -3.91769×10^-7, 3.4621×10^-7}
```

You may find it easier to do this in one step using **MapThread**.

```
MapThread[Times, mfsColumn[qpmfs, {"X", "PX"}]] // Short
{0., 0., 0., <<375>>, -3.91769×10^-7, 3.4621×10^-7}
```

7.2 Selection within Mfs objects

Many of the tables saved by MAD take the form of a list of machine elements together with the values of various optical functions (e.g., the OPTICS, TWISS3, ENVELOPE tables). Although MAD itself provides a selection mechanism for limiting these tables to certain classes of elements, it is often convenient to select further after the optics calculations have been done. You might want to simply tabulate the functions at some known subset of the elements originally chosen. Or, more interestingly, you might want to select elements according to the *values of the optical functions*. For example, elements with large β functions might be potential aperture limits.

In the following we provide a general selection mechanism and some simpler ones for common cases.

7.2.1 An OPTICS table example

Create an **mfs** data object from an OPTICS table containing values at the IPs and pickups in LEP:

```
opticsfile =
 ToFileName[{AfsHomeDirectory["jowett"], "public", "math",
   "Madtomma", "Mfs", "Examples"}, "OPTICSdp0000.tfs"]
P:\cern.ch\user\j\jowett\public\
  math\Madtomma\Mfs\Examples\OPTICSdp0000.tfs
```

```
OPTICS[0] = tfsRead[opticsfile, Verbose -> False];
```

```
mfsColumnNames[OPTICS[0]]
{NAME, BETX, DX, BETY}
```

7.2.2 General Selection Method

mfsSelect is a very general selection function modelled on the standard *Mathematica* function **Select** [1]. To understand how it works, you should first have some understanding of **Select** and the notion of pure functions in *Mathematica*. If you do not, then I recommend that you this subsection and look at the simpler selection functions (**mfsMember**, **mfsRange**) below. They cover many practical cases.

```
?mfsSelect
```

```
mfsSelect[mfsdata,criterion] extracts rows
  satisfying criterion (function) from an mfsdata object.
```

mfsSelect is usually used in conjunction with an auxiliary function **mfsColumnValue.** This allows selection from a specified row. To see how it works, we can look at the row number 14 in the OPTICS table. This function can be used to extract the value of the column **"NAME"** for that row.

```
arow = mfsColumn[OPTICS[0]][[14]]
{PU.QL15.R1, 129.71, 0.987151, 21.1482}
```

```
mfsColumnValue[OPTICS[0], arow, "NAME"]
PU.QL15.R1
```

Now the **mfsColumnValue** function allows a very convenient specification of a selection criterion, . For example, using it to construct an appropriate pure function, you can create a *new* **mfs** data object containing only the elements where $\beta_x > 250$:

```
hibetx = mfsSelect[OPTICS[0],
   (mfsColumnValue[OPTICS[0], #, "BETX"] > 250.) &];
```

Then we can make a table showing the values of both β_x and β_y at these elements:

```
Transpose[mfsColumn[hibetx, {"NAME", "BETX", "BETY"}]] // TableForm
PU.QS1A.L2      294.131      70.5863
PU.QS1A.R2      294.136      70.5862
PU.QS1A.L4      321.516      74.436
PU.QS1A.R4      321.519      74.4358
PU.QS1A.L6      299.147      76.797
PU.QS1A.R6      299.145      76.7969
PU.QS1A.L8      321.465      74.4562
PU.QS1A.R8      321.467      74.4614
```

As a further example, we can select all the rows whose name contains the string **QL11.** The similarity to the mechanism used in **Select** should now be very obvious.

```
TableForm[Transpose[mfsColumn[mfsSelect[OPTICS[0],
    StringMatchQ[mfsColumnValue[OPTICS[0], #1, "NAME"],
      "*QL11*"] &], {"NAME", "BETX", "DX"}]]]
PU.QL11.R1      146.035      7.4742×10^-9
PU.QL11.L3      146.041      -2.42813×10^-7
PU.QL11.R3      146.041      2.34022×10^-7
PU.QL11.L5      146.042      2.01702×10^-7
PU.QL11.R5      146.042      -1.78043×10^-7
PU.QL11.L7      146.041      -1.20652×10^-7
PU.QL11.R7      146.041      8.66908×10^-8
PU.QL11.L1      146.036      2.46937×10^-8
```

Arbitrary logical cominations of criteria can be built up using standard *Mathematica* functions and operators. Here, we select those elements containing **"QS1"** in their names where $\beta_x > 320$:

```
mfsSelect[OPTICS[0],
 StringMatchQ[mfsColumnValue[OPTICS[0], #1, "NAME"], "*QS1*"] &&
   mfsColumnValue[OPTICS[0], #1, "BETX"] > 320. &]
mfs[{{GAMTR, 73.4237}, {ALFA, 0.000185493},
  {XIY, 0.987098}, {XIX, 1.00236}, {QY, 76.1942}, {QX, 90.2859},
  {CIRCUM, 26658.872082}, {DELTA, 0.}, {TYPE, OPTICS},
  {COMMENT, N0520P97v2.lep, 90/60 BT optics for 1997 up to 97 GeV},
  {ORIGIN, MAD 8.21/13 HP/UX}, {DATE, 12/08/97},
  {TIME, 15.04.39}}, {NAME, BETX, DX, BETY},
 {{PU.QS1A.L4, 321.516, -2.47388×10^-7, 74.436},
  {PU.QS1A.R4, 321.519, 6.39189×10^-8, 74.4358},
  {PU.QS1A.L8, 321.465, -3.40013×10^-7, 74.4562},
  {PU.QS1A.R8, 321.467, 3.03216×10^-7, 74.4614}}]
```

If you know the internal structure of the mfs data object, you can refer directly to its elements without using the **mfsColumnValue** function directly (but this is not normally recommended). For instance, knowing the columns 2 and 2 contain the appropriate quantities you can select the rows where $\beta_x D_x > 140$:

```
result = mfsSelect[OPTICS[0], #1[[2]] #1[[3]] > 140. &];
```

```
Transpose[mfsColumn[result, {"NAME", "BETX", "DX"}]] // TableForm
PU.QS15.L2      149.927      0.950618
PU.QS15.R2      149.922      0.950618
```

This is equivalent to the selection

```
result1 = mfsSelect[OPTICS[0], (mfsColumnValue[OPTICS[0], #, "BETX"]
       mfsColumnValue[OPTICS[0], #, "DX"] > 140.) &];
```

7.2.3 Simple selection mechanisms

Common cases can be handled with simpler functions, e.g., when the names in a given column belongs to a specified set or numerical values fall in a given range.

To select from a set, we can first create a set of interaction point names and then select all the rows containing them.

```
IPs = {"IP2", "IP4", "IP6", "IP8"};
```

```
IPoptics = mfsMember[OPTICS[0], "NAME", IPs]
mfs[{{GAMTR, 73.4237}, {ALFA, 0.000185493},
  {XIY, 0.987098}, {XIX, 1.00236}, {QY, 76.1942}, {QX, 90.2859},
  {CIRCUM, 26658.872082}, {DELTA, 0.}, {TYPE, OPTICS},
  {COMMENT, N0520P97v2.lep, 90/60 BT optics for 1997 up to 97 GeV},
  {ORIGIN, MAD 8.21/13 HP/UX}, {DATE, 12/08/97}, {TIME, 15.04.39}},
 {NAME, BETX, DX, BETY}, {{IP2, 2., 2.87934×10^-8, 0.0499991},
  {IP4, 2.00002, -2.57651×10^-8, 0.0499995},
  {IP6, 2.00002, 1.76834×10^-8, 0.0500001},
  {IP8, 1.99995, -5.16881×10^-9, 0.0499975}}]
```

```
Transpose[mfsColumn[IPoptics, {"NAME", "BETX", "BETY"}]] // TableForm
IP2     2.          0.0499991
IP4     2.00002     0.0499995
IP6     2.00002     0.0500001
IP8     1.99995     0.0499975
```

A similar function handles the case when a value is a number in a specified range. For instance, select the rows with $120 < \beta_x < 126$:

```
betxval = mfsRange[OPTICS[0], "BETX", {120, 126}] ;
```

```
Transpose[mfsColumn[betxval, {"NAME", "BETX"}]] // TableForm
PU.QL2B.R1      125.212
PU.QL2B.L3      125.213
PU.QL2B.R3      125.21
PU.QL2B.L5      125.212
PU.QL2B.R5      125.211
PU.QL2B.L7      125.21
PU.QL2B.R7      125.213
PU.QL2B.L1      125.209
```

Of course, the limits of the range need not be finite. To find all places with $\beta_x > 300$, we can use

```
hibetx = mfsRange[OPTICS[0], "BETX", {300, ∞}] ;
```

```
mfsColumn[hibetx, "NAME"]
{PU.QS1A.L4, PU.QS1A.R4, PU.QS1A.L8, PU.QS1A.R8}
```

7.2.4 Other Selection Methods

mfsSelect is quite general and useful. However other methods of data selection are also possible. For instance, you can use BooleanSelect from the Statistics‘DataManipulation‘ package (which is made available by the **Mfs** package). As an example, first construct a boolean array that indicates all places where the product $\beta_x D_x > 120$.

```
test = Greater[#, 120.] & /@
   ( mfsColumn[OPTICS[0], "BETX"] mfsColumn[OPTICS[0], "DX"] );
```

```
Short[test, 5]
{False, False, False, False, False, False, False, False,
 False, False, False, False, False, True, False, True, False,
 False, False, False, False, False, False, False, False, False,
 <<457>>, False, False, False, False, False, False, False, False,
 False, False, True, False, True, False, False, False, False,
 False, False, False, False, False, False, False, False, False}
```

Then get the rows where the product $\beta_x D_x > 120$.

```
?BooleanSelect
```

```
BooleanSelect[list, sel] keeps elements of list
  for which the corresponding element of sel is True.
```

```
BooleanSelect[
 Transpose[ mfsColumn[OPTICS[0], {"NAME", "BETX", "DX"}]], test]
{{PU.QL15.R1, 129.71, 0.987151}, {PU.QL17.R1, 117.425, 1.07893},
 {PU.QS15.L2, 149.927, 0.950618}, {PU.QS15.R2, 149.922, 0.950618},
 {PU.QL17.L3, 117.415, 1.07886}, {PU.QL15.L3, 129.783, 0.987148},
 {PU.QL15.R3, 129.783, 0.987147}, {PU.QL17.R3, 117.418, 1.07886},
 {PU.QS15.L4, 139.083, 0.891688}, {PU.QS15.R4, 139.082, 0.891688},
 {PU.QL17.L5, 117.417, 1.07886}, {PU.QL15.L5, 129.783, 0.987147},
 {PU.QL15.R5, 129.783, 0.987148}, {PU.QL17.R5, 117.417, 1.07886},
 {PU.QS17.L6, 118.596, 1.10177}, {PU.QS17.R6, 118.596, 1.10177},
 {PU.QL17.L7, 117.418, 1.07886}, {PU.QL15.L7, 129.783, 0.987148},
 {PU.QL15.R7, 129.783, 0.987147}, {PU.QL17.R7, 117.415, 1.07886},
 {PU.QS15.L8, 139.082, 0.891688}, {PU.QS15.R8, 139.088, 0.891688},
 {PU.QL17.L1, 117.428, 1.07893}, {PU.QL15.L1, 129.71, 0.987151}}
```

Still another way to do this is to use **MapThread** to form the product of the two columns together with the element names. Then the elements can be selected.

```
Select[MapThread[{#1, #2 #3} &,
  mfsColumn[OPTICS[0], {"NAME", "BETX", "DX"}]], Last[#1] > 120. &]
{{PU.QL15.R1, 128.043}, {PU.QL17.R1, 126.693},
 {PU.QS15.L2, 142.523}, {PU.QS15.R2, 142.519},
 {PU.QL17.L3, 126.674}, {PU.QL15.L3, 128.115},
 {PU.QL15.R3, 128.115}, {PU.QL17.R3, 126.678},
 {PU.QS15.L4, 124.019}, {PU.QS15.R4, 124.018},
 {PU.QL17.L5, 126.677}, {PU.QL15.L5, 128.115},
 {PU.QL15.R5, 128.115}, {PU.QL17.R5, 126.677},
 {PU.QS17.L6, 130.666}, {PU.QS17.R6, 130.666},
 {PU.QL17.L7, 126.678}, {PU.QL15.L7, 128.115},
 {PU.QL15.R7, 128.115}, {PU.QL17.R7, 126.674},
 {PU.QS15.L8, 124.018}, {PU.QS15.R8, 124.023},
 {PU.QL17.L1, 126.697}, {PU.QL15.L1, 128.043}}
```

7.3 Modifying and saving mfs data objects

7.3.1 Modifying the header part of the object

It may be useful to add additional keys and values that were not given in the original TFS file. These can be any kind of object definable in *Mathematica*.

```
qpmfs =
  mfsAddKey[qpmfs, {"REMARK", "Example for conference paper."}];
```

```
qpmfs = mfsAddKey[qpmfs, {"KICKVALUE", 4 × 10^-3}];
```

```
qpmfs = mfsAddKey[qpmfs, {"AMATRIX", Inverse[( σ   α  )]}];
                                              ( β  -σ^-1 )
```

```
mfsKeyValue[qpmfs, "AMATRIX"] // MatrixForm
```

$$\begin{pmatrix} -\frac{1}{(-1-\alpha\,\beta)\,\sigma} & -\frac{\alpha}{-1-\alpha\,\beta} \\ -\frac{\beta}{-1-\alpha\,\beta} & \frac{\sigma}{-1-\alpha\,\beta} \end{pmatrix}$$

Descriptors can be changed with **mfsEditKey** as follows. In this example, the **COMMENT** is empty since nothing was supplied in the MAD run:

```
mfsKeyValue[qpmfs, "COMMENT"]
```

However it can be added as an afterthought:

```
qpmfs =
  mfsEditKey[qpmfs, {"COMMENT", "Tracking from an initial kick."}];
```

```
mfsKeyValue[qpmfs, "COMMENT"]
Tracking from an initial kick.
```

Descriptors can be deleted with

```
qpmfs = mfsDeleteKey[qpmfs, "ET"];
```

The list of keys now reads

```
mfsKeyNames[qpmfs]
{TYPE, X, PX, Y, PY, EY, EX, E11, E12, E13, E14, E15, E16, E21, E22, E23,
 E24, E25, E26, E31, E32, E33, E34, E35, E36, E41, E42, E43, E44,
 E45, E46, E51, E52, E53, E54, E55, E56, E61, E62, E63, E64, E65,
 E66, COMMENT, ORIGIN, DATE, TIME, REMARK, KICKVALUE, AMATRIX}
```

7.3.2 Adding and removing columns

Let us show how to add another column of data to the mfs data object **OPTICS[0]**. An additional column should have length

```
collen = Length[Last[OPTICS[0]]]
509
```

Let us construct a simple index of the elements since this particular data object contains no azimuth "S"

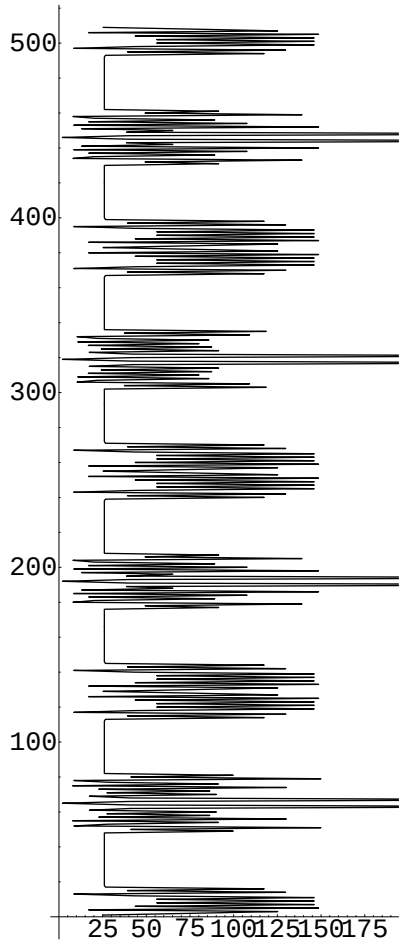
```
idx = Range[collen] ;
```

```
OPTICS[1] = mfsAddColumn[OPTICS[0], "INDEX", idx];
```

Now there is a new column that we can use in a trivial way to produce a plot

```
mfsColumnNames[OPTICS[1]]
{BETX, BETY, DX, INDEX, NAME}
```

```
ListPlot[Transpose[mfsColumn[OPTICS[1], {"BETX", "INDEX"}]],
 PlotJoined → True, AspectRatio -> Automatic]
```




```
- Graphics -
```

There are also functiosn for removing keys and columns

```
nodx = mfsDeleteKey[
        mfsDeleteColumn[OPTICS[0], "DX"],
            "DELTA"];
```

```
nobetas = mfsDeleteColumn[OPTICS[0], {"BETX", "BETY"}];
```

```
mfsKeyNames[nodx]
{GAMTR, ALFA, XIY, XIX, QY, QX,
 CIRCUM, TYPE, COMMENT, ORIGIN, DATE, TIME}
```

```
mfsColumnNames[nodx]
{NAME, BETX, BETY}
```

```
mfsColumnNames[nobetas]
{NAME, DX}
```

We shall use these reduced data objects in the next subsection.

7.3.3 Merging several mfs objects and making sure they match

It is often useful to be able to combine the contents of two or more table files generated by MAD. For example you may want to combine SURVEY and TWISS information for the same set of elements. The function **mfsMerge** allows you to combine two mfs objects. Using it, we can recombine the two partial optics tables

```
togetherAgain = mfsMerge[{nodx, nobetas}];
```

```
mfsColumnNames[togetherAgain]
{BETX, BETY, DX, NAME}
```

The columns of data in the mfs objects must be of the same length. If column names are duplicated in more than one of the mfs objects only the column belonging to the first mfs object in the list is retained.

In addition, the function **mfsMerge** has an option designed to ensure that columns that are duplicated in more than one of the mfs objects are identical or, in the case of numerical data, equal to machine precision. By default only the columns called "NAME" are checked.

```
Options[mfsMerge]
{matchColumns → {NAME}}
```

To enforce the stronger requirement, that all columns with identical names match up, do

```
SetOptions[mfsMerge, matchColumns -> Automatic]
{matchColumns → Automatic}
```

or give the option on individual calls to **mfsMerge**.

7.3.4 Reversing the order in the columns

In MAD, calculations for electrons have to be done with the sequence of elements reversed with respect to that for positrons. So we need a function to reverse the order of the rows in the block of columns :

```
Short[mfsColumn[mfsReverse[ OPTICS[0]], "NAME"], 20]
{ENDLEP, PU.QL1B.L1, PU.QL2B.L1, PU.QL4B.L1, PU.QL5.L1,
 PU.QL6.L1, PU.QL7.L1, PU.QL8.L1, PU.QL9.L1, PU.QL10.L1,
 PU.QL11.L1, PU.QL12.L1, PU.QL14.L1, PU.QL15.L1, PU.QL16.L1,
 PU.QL17.L1, PU.QL18.L1, PU.QD20.L1, PU.QD22.L1, PU.QD24.L1,
 PU.QD26.L1, PU.QD28.L1, PU.QD30.L1, PU.QD32.L1, PU.QD34.L1,
 PU.QD36.L1, PU.QD38.L1, PU.QD40.L1, PU.QD42.L1, PU.QD44.L1,
 PU.QD46.L1, PU.QD48.L1, PU.QD48.R8, PU.QD46.R8, PU.QD44.R8,
 PU.QD42.R8, PU.QD40.R8, PU.QD38.R8, PU.QD36.R8, PU.QD34.R8,
 PU.QD32.R8, PU.QD30.R8, PU.QD28.R8, PU.QD26.R8, PU.QD24.R8,
 PU.QD22.R8, PU.QD20.R8, PU.QS18.R8, PU.QS17.R8, PU.QS16.R8,
 <<410>>, PU.QS17.L2, PU.QS18.L2, PU.QD20.L2, PU.QD22.L2,
 PU.QD24.L2, PU.QD26.L2, PU.QD28.L2, PU.QD30.L2, PU.QD32.L2,
 PU.QD34.L2, PU.QD36.L2, PU.QD38.L2, PU.QD40.L2, PU.QD42.L2,
 PU.QD44.L2, PU.QD46.L2, PU.QD48.L2, PU.QD48.R1, PU.QD46.R1,
 PU.QD44.R1, PU.QD42.R1, PU.QD40.R1, PU.QD38.R1, PU.QD36.R1,
 PU.QD34.R1, PU.QD32.R1, PU.QD30.R1, PU.QD28.R1, PU.QD26.R1,
 PU.QD24.R1, PU.QD22.R1, PU.QD20.R1, PU.QL18.R1, PU.QL17.R1,
 PU.QL16.R1, PU.QL15.R1, PU.QL14.R1, PU.QL12.R1, PU.QL11.R1,
 PU.QL10.R1, PU.QL9.R1, PU.QL8.R1, PU.QL7.R1, PU.QL6.R1,
 PU.QL5.R1, PU.QL4B.R1, PU.QL2B.R1, PU.QL1B.R1, IP1}
```

Other changes to the columns of data have not been implemented so far.

7.3.5 Saving an mfs object in a file

The new version of the object can then be saved in a file specified by the user. Here we take a temporary file.

```
junk = OpenTemporary[];
Save[junk, qpmfs];
Close[junk]
C:\TEMP\000003a00197
```

Of course, a file can contain any number of mfs objects together with other *Mathematica* objects.

7.3.6 Generating new variables

Sometimes it is useful to use the descriptors to create new symbols in the current context. This works as follows. The **Mfs** package contains a function which creates a symbol from a string and assigns it a value.

```
?interpretTagValue
```

```
interpretTagValue[{"tag",val}] creates
  a variable tag and assigms it the value val.
```

```
interpretTagValue[{"abc", 123}]
123
```

The symbol now evaluates

```
abc
123
```

This is used by the following function to create symbols for all the keys in an mfs data object.

```
interpretDescriptors[qpmfs];
```

Now we can evaluate them or use them in calculations by referring to their names.

```
{TYPE, KICKVALUE/Sqrt[EX + EY]}
{TRACK, 23.1206}
```

7.4 Updating files created with earlier versions of the package

Some functions names were changed in Version 2.0. The following set of rules gives the correspondences.

```
mfsVersion2UpdateRules
{interpretDescriptors → mfsInterpretKeys, keyValue → mfsKeyValue,
 columnNames → mfsColumnNames, descriptorNames → mfsKeyNames,
 colValue → mfsColumnValue, addDescriptor → mfsAddKey,
 editDescriptor → mfsEditKey, deleteDescriptor → mfsDeleteKey}
```

In fact, the old names still work but generate warning messages. To update notebooks and other files based on older versions of the package you can use the function

```
?mfsVersion2Update
```

```
mfsVersion2Update[oldfile,newfile] will create a
  new version of a file in which names of functions
  in Version 1.x of the Madtomma‘Mfs‘Mfs‘ package
  are changed to their equivalents in Version 2.x.
```

7.5 Extra functionality and utilities

8 Palettes for this package

Mathematica's palettes provide a convenient way to input commands without having to remember their names or syntax. They also help considerably to reduce syntax errors when you build up complex expressions. By selecting it ant then using the **File/Generate Palette from Selection** command, the following input cell can be transformed into a palette containing templates for the most commonly used commands in this package.

```
Needs["Madtomma`Mfs`Mfs`"]
tfsRead[■]
mfsKeyNames[■]
mfsKeyValue[■, □]
mfsColumnNames[■]
mfsColumn[■, "□"]
mfsColumn[■, {"□", "□"}]
mfsMember[■, "□", "□"]
mfsRange[■, "□", {□, □}]
mfsInterpretKeys[■]
mfsSelect[■, □]
mfsColumnValue[■, □, "□"]
```

The option **ShowGroupOpenCloseIcon->True** is useful in palettes. Note that the palette buttons are most useful if you first select a part of an expression on your screen to which you want to apply them.

Generating your own palette from this notebook has the advantage that you can modify it to your taste. Otherwise, the default ready-made palette will appear automatically on your screen when you load the package (if you don't want it, just close it).

Of course, you can also generate commands quickly by other standard mechanisms such as auto-completion (Ctrl-K or Ctrl-Shift-K). As in all properly constructed packages, every function has an associated usage message giving brief instructions, e.g.,

```
?interpretDescriptors
```

```
Obsolete function name.  Please use mfsInterpretKeys.
```

Here is another palette for the functions used to modify mfs objects. It is available as **MfsEditPalette.nb.**

```
mfsAddKey[■, {□, □}]
mfsEditKey[■, {□, □}]
mfsDeleteKey[■, □]
mfsAddColumn[■, □, □]
mfsDeleteColumn[■, □]
mfsMerge[{■, □}]
mfsReverse[■]
```

9 Bibliography

[1] Stephen Wolfram, *The Mathematica Book*, 3rd ed., Wolfram Media/Cambridge University Press, 1996.

[2] Roman Maeder, *Programming in Mathematica*, 3rd ed. Addison-Wesley, 1996

http://www.wolfram.com/~maeder/ProgInMath/

[3] Hans Grote, F. Christoph Iselin, The MAD Program, User's Reference Manual, CERN/SL/90-13 (AP), and
http://wwwslap.cern.ch/~fci/mad/mad.html